AI-based Financial Portfolio Manager using DRL

Banoth Tharun Tej

Overview

This project is about building an AI-based system that learns how to manage a stock portfolio using Deep Reinforcement Learning (DRL). Instead of writing fixed rules to decide when to buy or sell stocks, I trained a model that learns by itself from real-world data. The model looks at past stock prices and decides how much to invest in each stock to make profits over time. I used the PPO (Proximal Policy Optimization) algorithm and real stock price data fetched using the `yfinance` library. The full system includes data extraction, environment setup, model training, and result evaluation.

1. Data Extraction and Preprocessing

The first step in the project was to get stock market data. I used the `yfinance` Python library to download daily price data for multiple companies like Apple (AAPL), Microsoft (MSFT), and others. I selected a fixed time period and collected the data for all stocks in a consistent format.

After downloading, I aligned the data so that all stocks had values on the same dates. Then I removed any rows with missing prices to avoid errors later. Once I had clean data, I converted the raw prices into returns. This is done using the log-return formula:

$$r_t = \log\left(\frac{P_t}{P_{t-1}}\right)$$

This step helps us measure how much a stock's price changes from one day to the next in percentage terms.

To help the learning process, I normalized all the features. This means I made sure the data had a mean of 0 and a standard deviation of 1. Normalizing helps the AI model learn faster and prevents it from getting confused by large numbers.

2. Environment Design: PortfolioEnv

Next, I created a custom environment called `PortfolioEnv` which acts as a simulation of the stock market. This environment is based on the OpenAI Gym format and allows the reinforcement learning model to interact with stock data.

The environment takes in the current prices and the agent's portfolio weights (how much is invested in each stock). The model's output is an action vector that tells how much to buy or sell of each stock. After applying this action, the environment calculates a reward based on the return of the portfolio.

I added features to the environment such as:

- Applying transaction costs (so the model doesn't trade too frequently).

- Clipping extreme values to avoid unrealistic results.
- Proper resetting of the environment at the start of each episode.

This environment simulates how a real investor would make decisions and helps the agent learn which actions lead to profits.

3. Model Training and Evaluation

Once the environment was ready, I trained a reinforcement learning agent using the PPO algorithm from the `stable-baselines3` library. PPO is a popular DRL algorithm that is stable and works well for many real-world problems.

The training process works as follows:

- The agent observes stock prices and decides how to allocate the portfolio.
- The environment calculates the reward based on the portfolio's performance.
- The agent updates its internal policy using PPO rules to perform better next time.

I trained the model using:

```
model = PPO("MlpPolicy", env, verbose=1)
model.learn(total_timesteps=100_000)
model.save("ppo_portfolio")
```

After training, I tested the model by loading it and running it on unseen data to see how well it performs. I tracked the returns, reward trends, and how the portfolio value changed over time.

Conclusion

This project helped me understand how to apply AI techniques to finance. I fetched real stock data, cleaned and transformed it, and created a custom simulation environment. Then I trained a DRL model that learns to invest intelligently. The agent was able to learn how to distribute money among multiple stocks to maximize long-term returns. This work can be extended further by testing other RL algorithms, using more stocks, or integrating it into a Streamlit dashboard for live tracking.